LECTURE 01

TOPOLOGICAL SORT

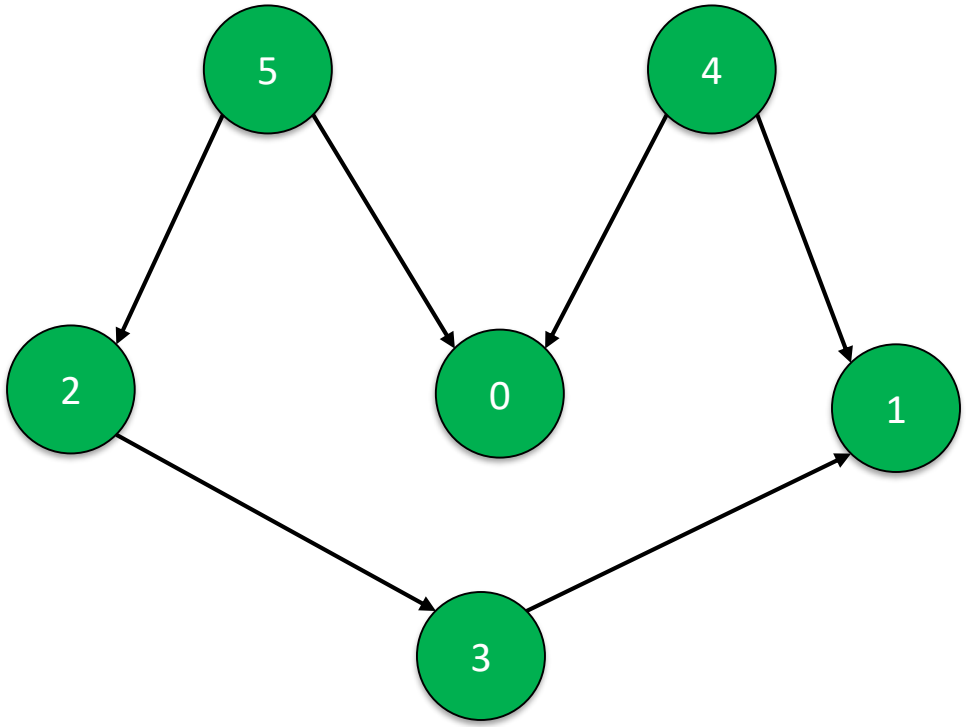
Big-O Coding

Website: www.bigocoding.com

Giới thiệu

Topological Sort (sắp xếp Topo): là một dạng sắp xếp các đỉnh trên đồ thị có hướng thành một dãy, sao cho mọi cung từ u đến v trong đồ thị, u luôn luôn nằm trước v.

- 5 đứng trước 2, 0
- 4 đứng trước 1, 0
- 2 đứng trước 3
- 3 đứng trước 1



Kết quả:

- 5, 4, 2, 3, 1, 0.
- 4, 5, 2, 0, 3, 1.

...

Một số lưu ý

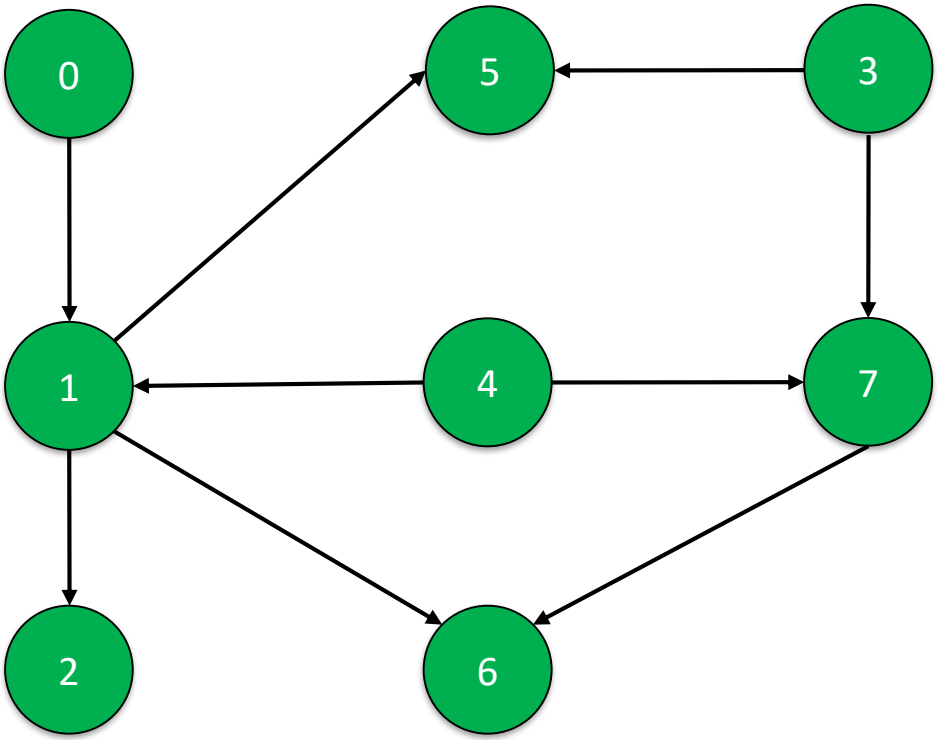
1. Thứ tự Topo tồn tại khi và chỉ khi đồ thị có hướng không có chu trình (DAG - Directed Acyclic Graph).
2. Đồ thị DAG có thể có nhiều thứ tự Topo.
3. Có 2 thuật toán để giải quyết bài toán Topological Sort:

- Thuật toán dựa trên ý tưởng DFS.
- Thuật toán Kahn (dựa trên ý tưởng BFS).

Độ phức tạp: O(V + E)

Giải bài toán bằng tay

Cho đồ thị như hình vẽ. Hãy sắp xếp các đỉnh đồ thị theo thứ tự Topo.



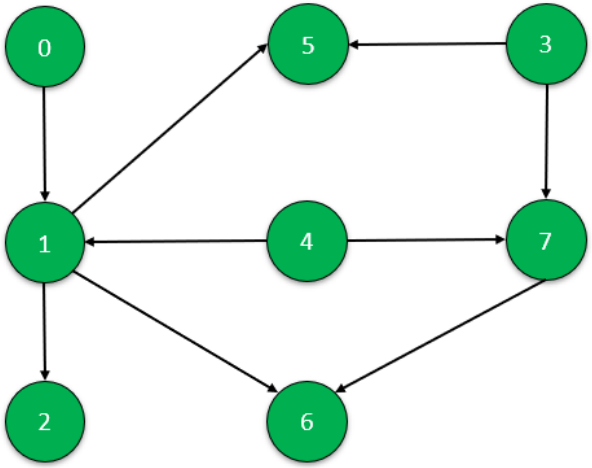
Edge List

```
8 9
0 1
1 2
1 5
1 6
4 1
4 7
7 6
3 7
3 5
```

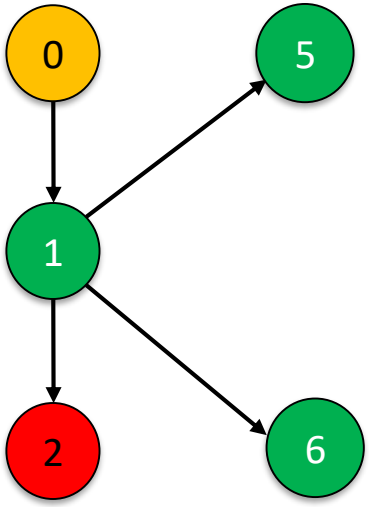
THUẬT TOÁN TOPO DỰA TRÊN DFS

Ý tưởng thuật toán

Từ danh sách đỉnh ban đầu, chọn 1 đỉnh chưa viếng thăm gọi DFS cho đỉnh này.



Từ đỉnh này đi xuống tới những đỉnh không có đỉnh kề nào.



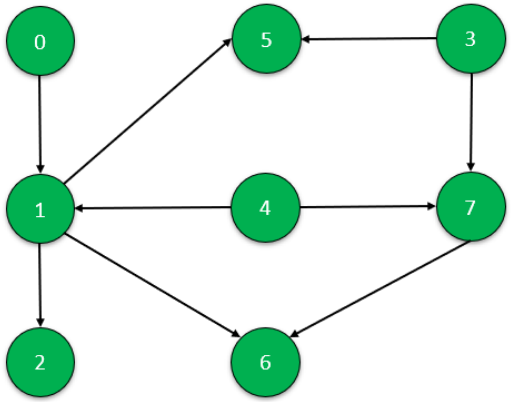
Tìm đỉnh mới để chạy DFS.

Lưu đỉnh tìm được vào mảng kết quả.

Quay lại đỉnh trước đó.

Vị trí	0	1	2	3
Đỉnh	2	...	...	...

Bước 0: Chuẩn bị dữ liệu (1)



Chuyển danh sách cạnh kề vào CTDL **graph**.

Đỉnh	0	1	2	3	4	5	6	7
Đỉnh kề	1	2, 5, 6	...	5, 7	1, 7	...	...	6

Mảng đánh dấu các đỉnh đã xét **visited**.

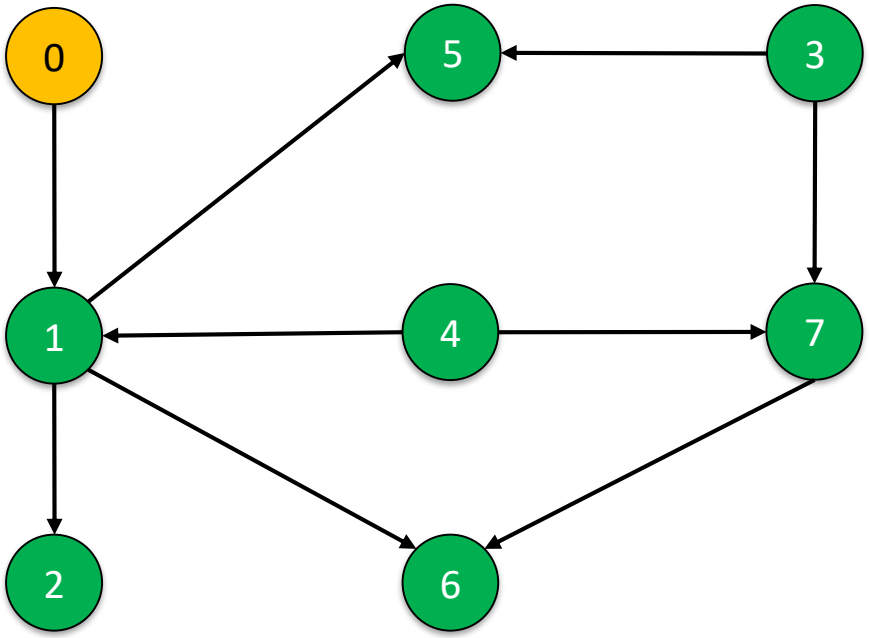
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	false	false	false	false	false	false	false	false

Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 0: Chuẩn bị dữ liệu (2)

Xét từng đỉnh trên đồ thị đỉnh nào chưa được viếng thăm sẽ gọi DFS đỉnh đó.



visited

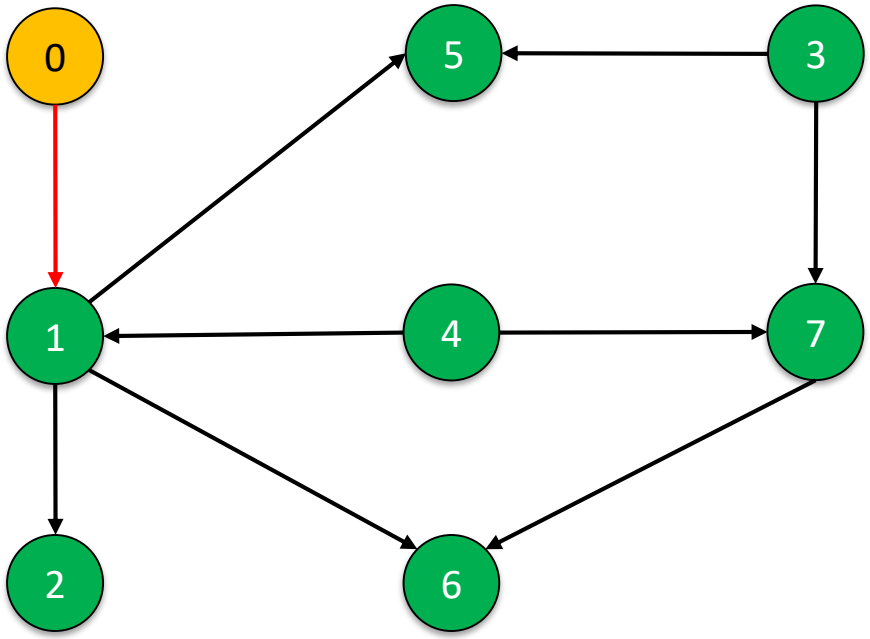
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	false	false	false	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 0

Gọi DFS đỉnh 0 → đi đến những đỉnh kề của đỉnh 0.



visited

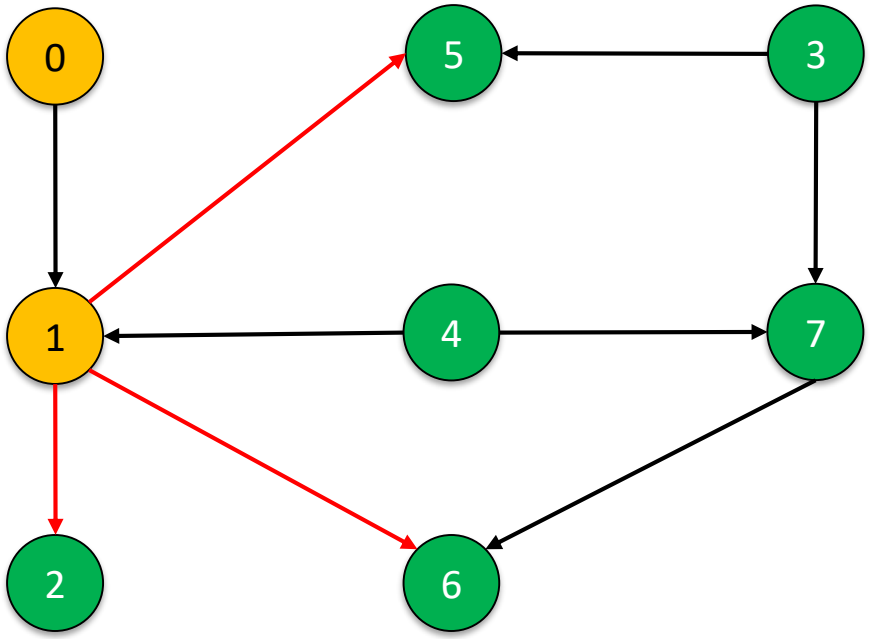
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	false	false	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 1

Gọi DFS đỉnh 1 → đi đến những đỉnh kề của đỉnh 1.



visited

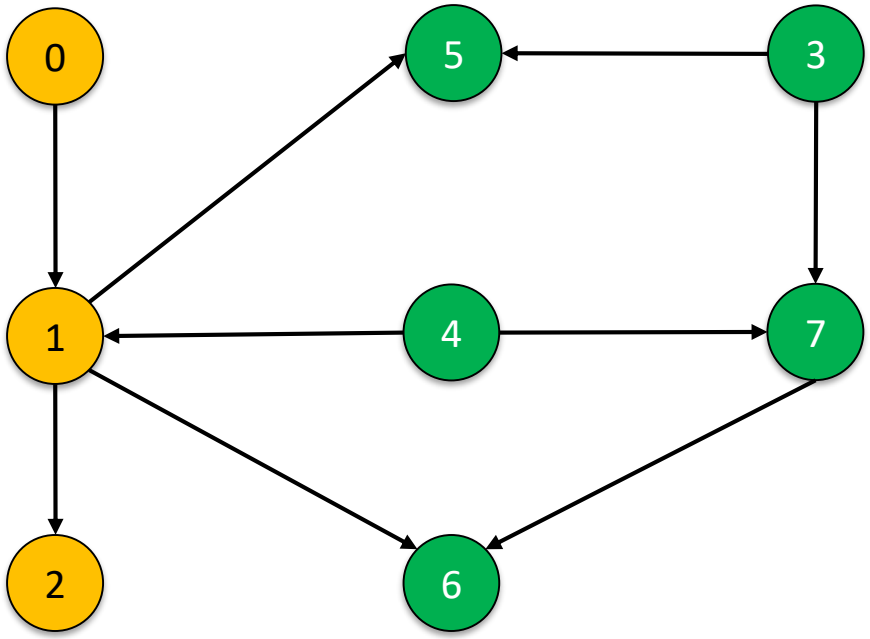
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	false	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 1: Chạy DFS đỉnh 2

Gọi DFS đỉnh 2 → đi đến những đỉnh kề của đỉnh 2.



visited

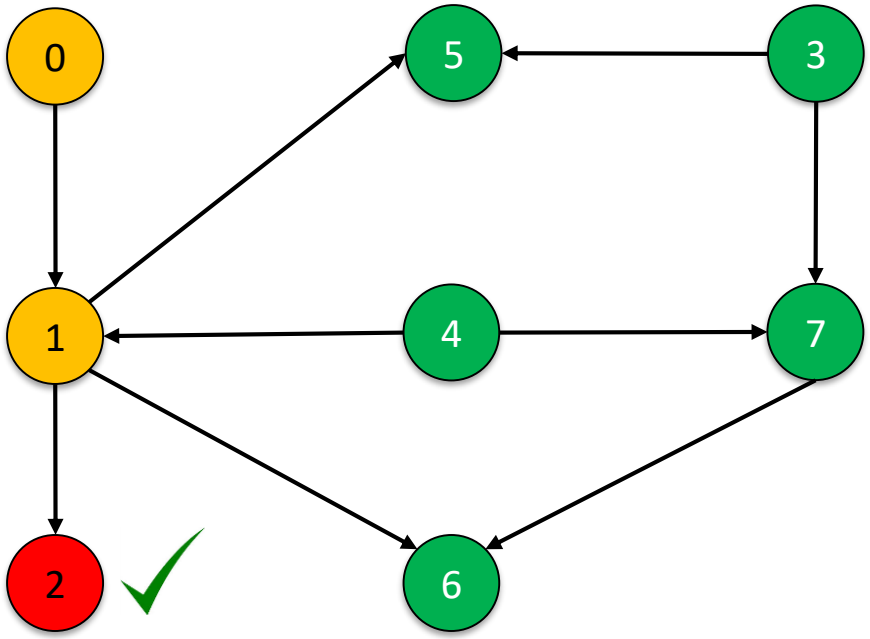
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 1: Lưu đỉnh 2 vào danh sách kết quả

Đỉnh 2 không còn đỉnh kề nào → lưu đỉnh 2 vào mảng kết quả.



visited

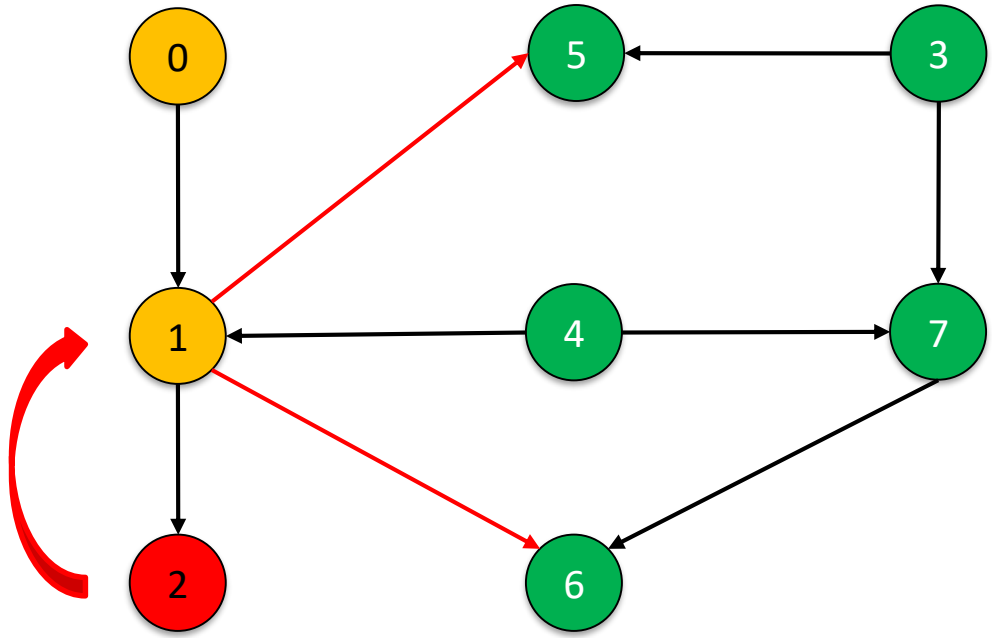
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	...	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 1 (back)

Gọi DFS đỉnh 1 → đi đến những đỉnh kề **tiếp theo** của đỉnh 1



visited

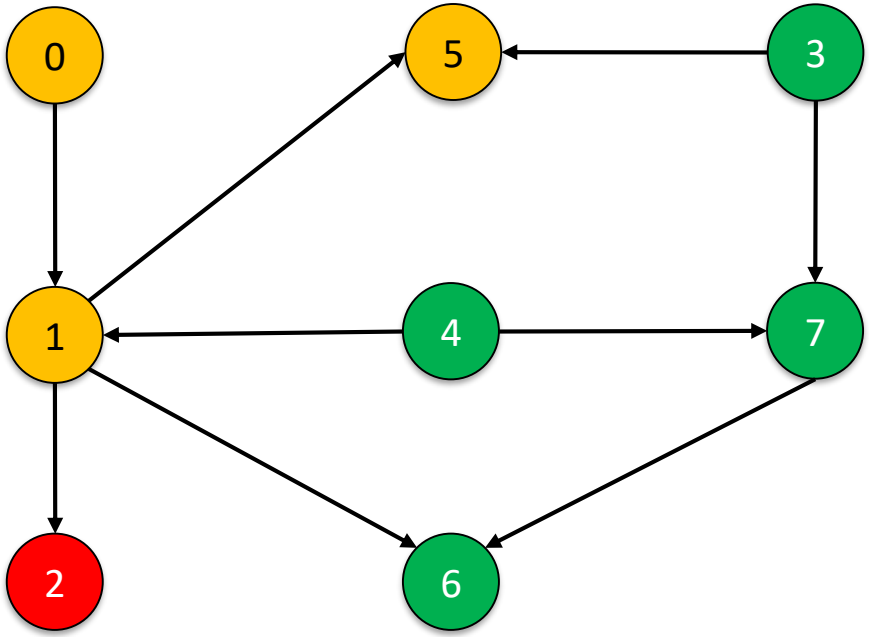
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	false	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	...	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 5

Gọi DFS đỉnh 5 → đi đến những đỉnh kề của đỉnh 5.



visited

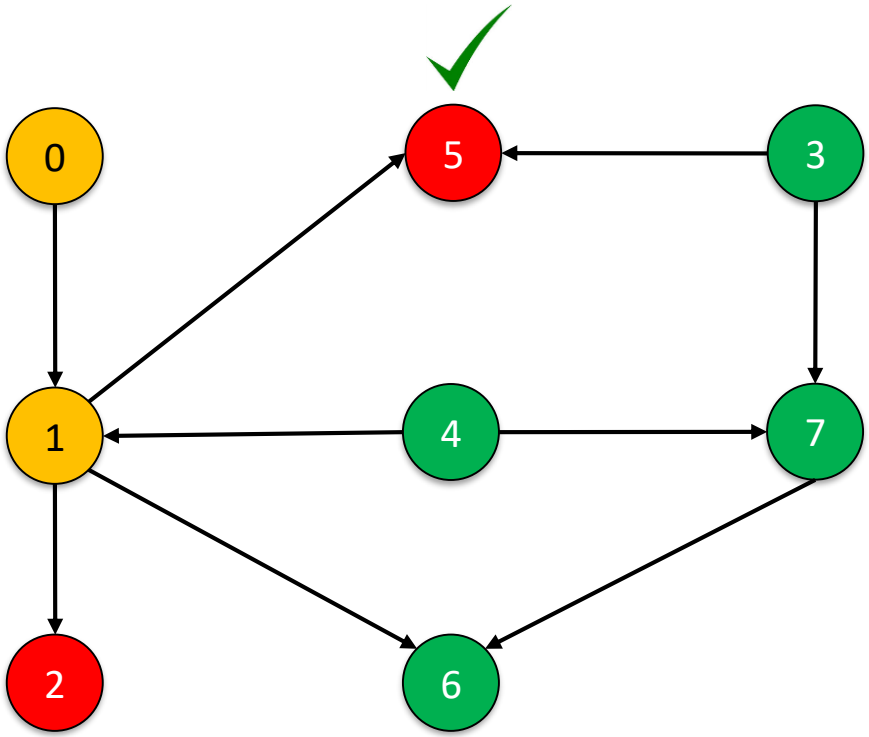
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	...	...	...	...	...	...	...

Bước 1: Lưu đỉnh 5 vào danh sách kết quả

Đỉnh 5 không còn đỉnh kề nào → lưu đỉnh 5 vào mảng kết quả.



visited

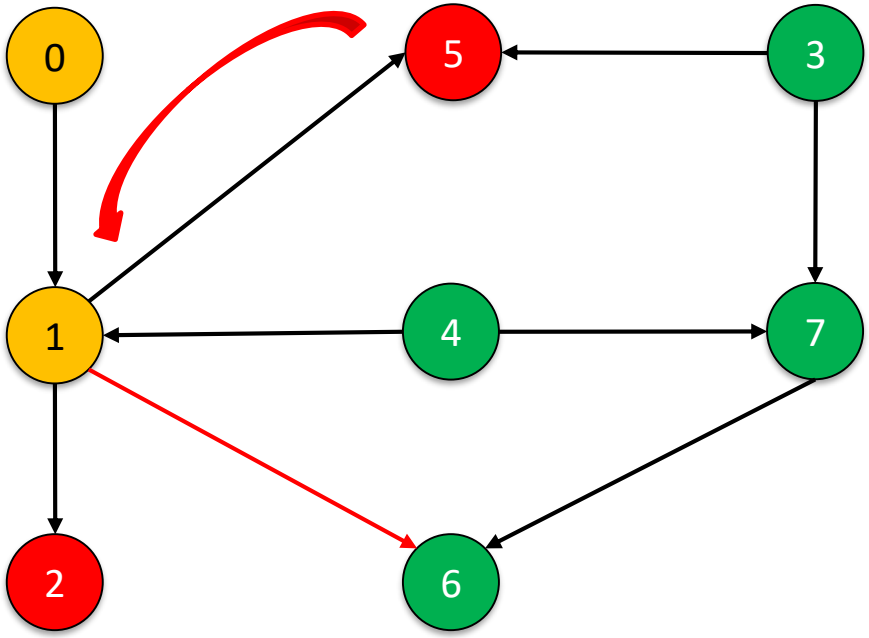
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 1 (back)

Gọi DFS đỉnh 1 → đi đến những đỉnh kề **tiếp theo** của đỉnh 1.



visited

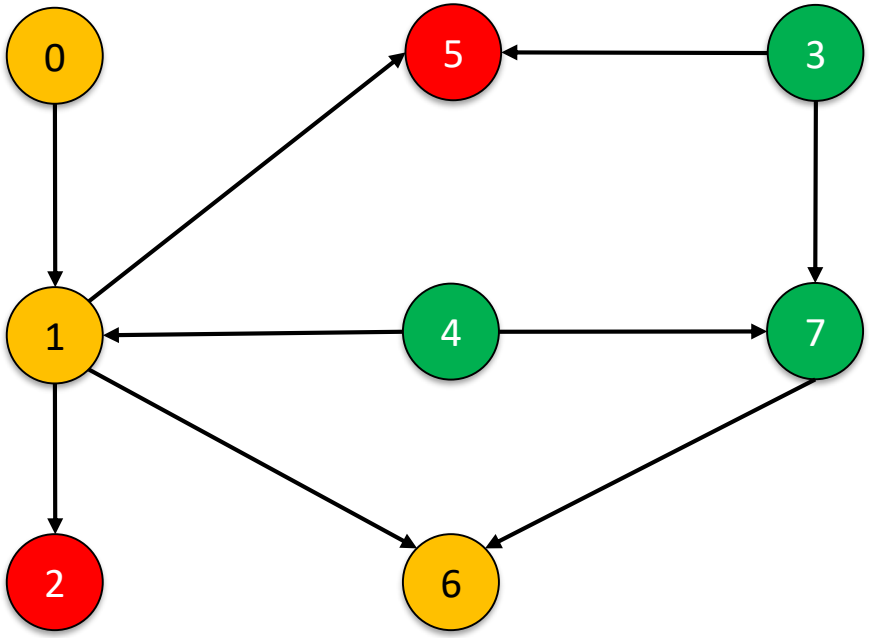
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	false	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	...	...	...	...	...	...

Bước 1: Chạy DFS cho đỉnh 6

Gọi DFS đỉnh 6 → đi đến những đỉnh kề **tiếp theo** của đỉnh 6.



visited

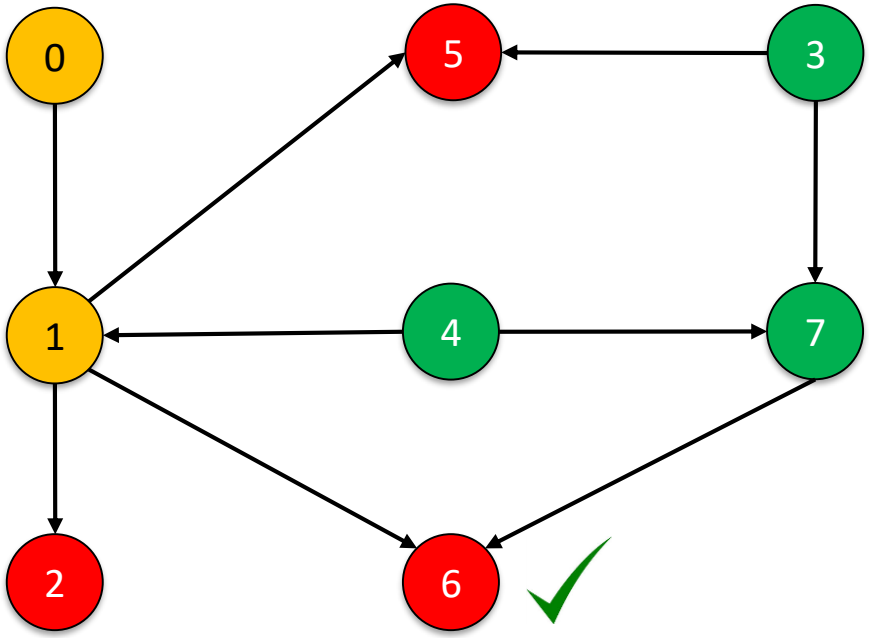
Đỉnh	0	1	2	3	4	5	**6**	7
Trạng thái	true	true	true	false	false	true	**true**	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	...	...	...	...	...	...

Bước 1: Lưu đỉnh 6 vào danh sách kết quả

Đỉnh 6 không còn đỉnh kề nào → lưu đỉnh 6 vào mảng kết quả.



visited

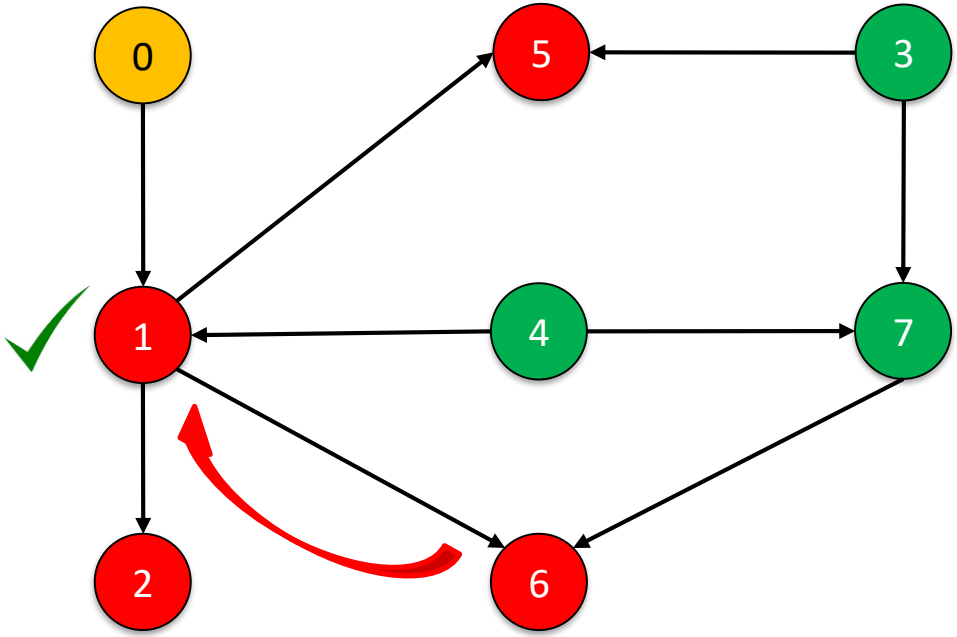
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	true	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	...	...	...	...	...

Bước 1: Lưu đỉnh 1 vào danh sách kết quả

Đỉnh 1 không còn đỉnh kề nào → lưu đỉnh 1 vào mảng kết quả.



visited

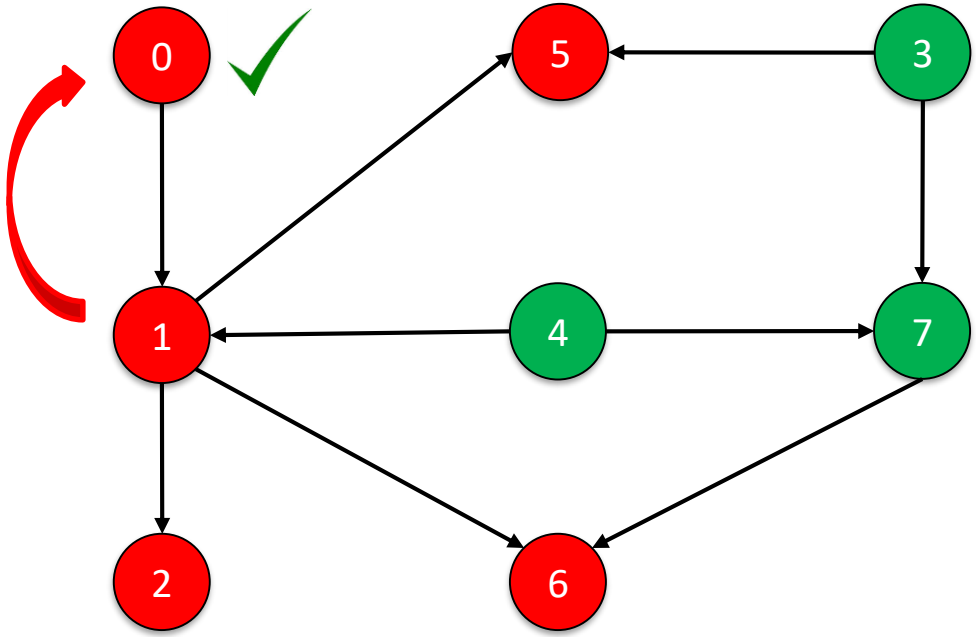
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	true	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	...	...	...	...

Bước 1: Lưu đỉnh 0 vào danh sách kết quả

Đỉnh 0 không còn đỉnh kề nào → lưu đỉnh 0 vào mảng kết quả.



visited

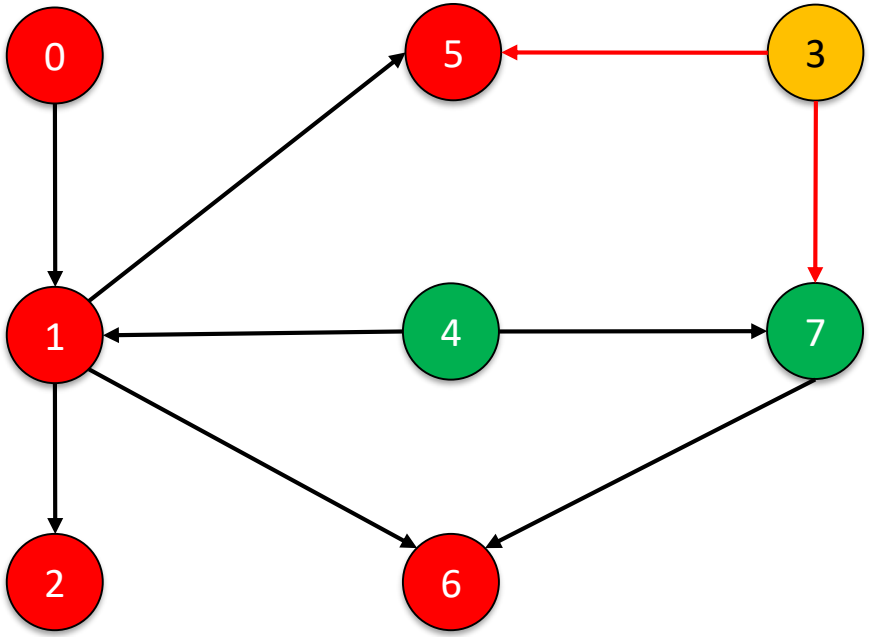
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	false	false	true	true	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	...	...	...

Bước 2: Chạy DFS cho đỉnh 3

Tìm đỉnh khác chưa viếng thăm. Gọi DFS đỉnh 3 → đi đến những đỉnh kề của đỉnh 3.



visited

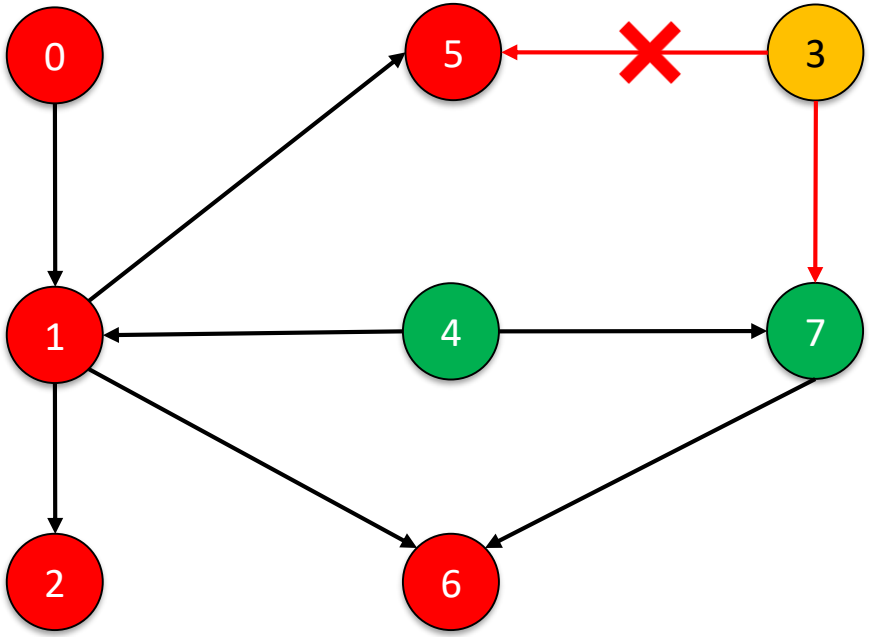
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	...	...	...

Bước 2: Chạy DFS cho đỉnh 5

Gọi DFS cho đỉnh 5, nhưng đỉnh 5 đã viếng thăm rồi → bỏ qua.



visited

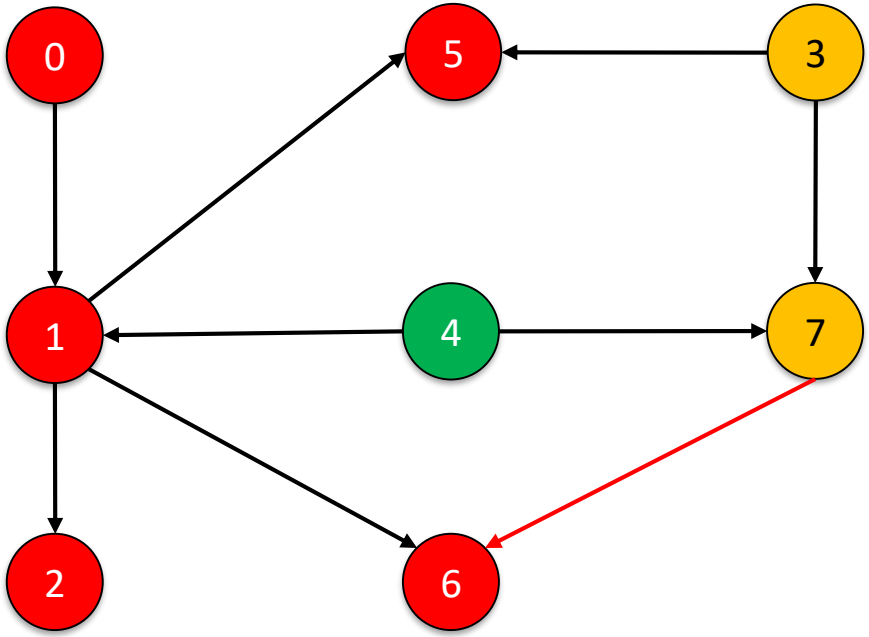
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	false

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	...	...	...

Bước 2: Chạy DFS cho đỉnh 7

Gọi DFS đỉnh 7 → đi đến những đỉnh kề của đỉnh 7.



visited

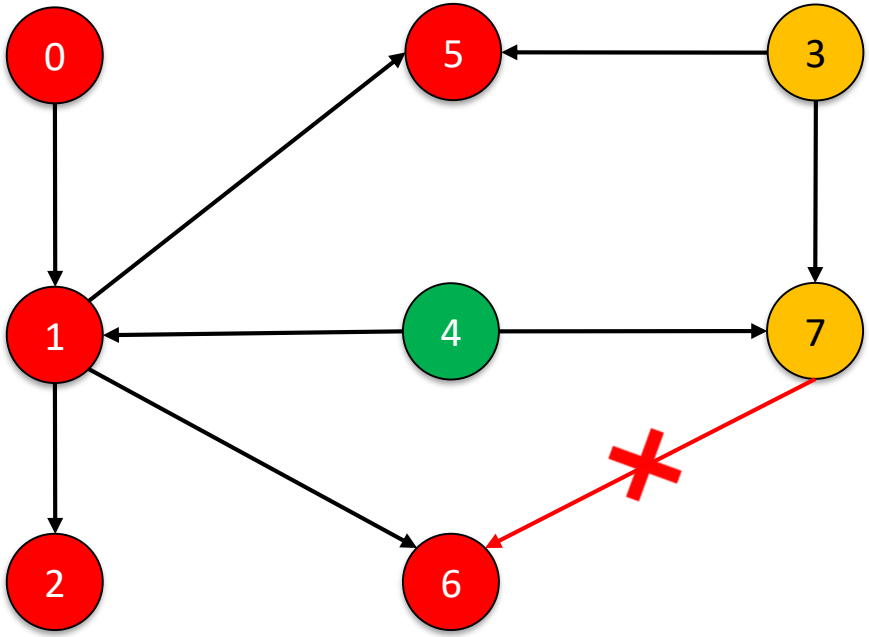
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	...	...	...

Bước 2: Chạy DFS cho đỉnh 6

Gọi DFS cho đỉnh 6, nhưng đỉnh 6 đã viếng thăm rồi → bỏ qua.



visited

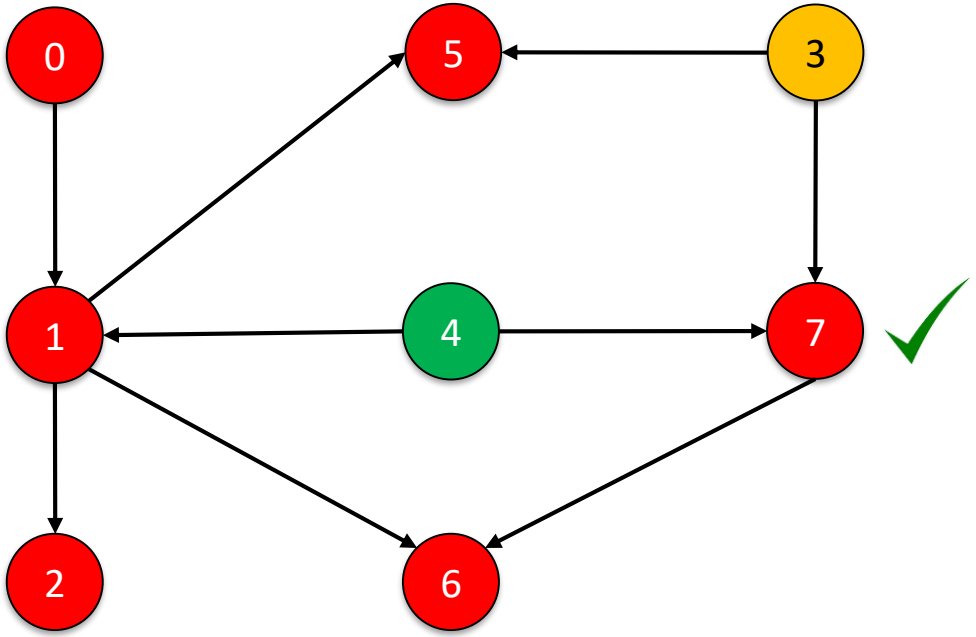
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	...	...	...

Bước 2: Lưu đỉnh 7 vào danh sách kết quả

Đỉnh 7 không còn đỉnh kề nào → lưu đỉnh 7 vào mảng kết quả.



visited

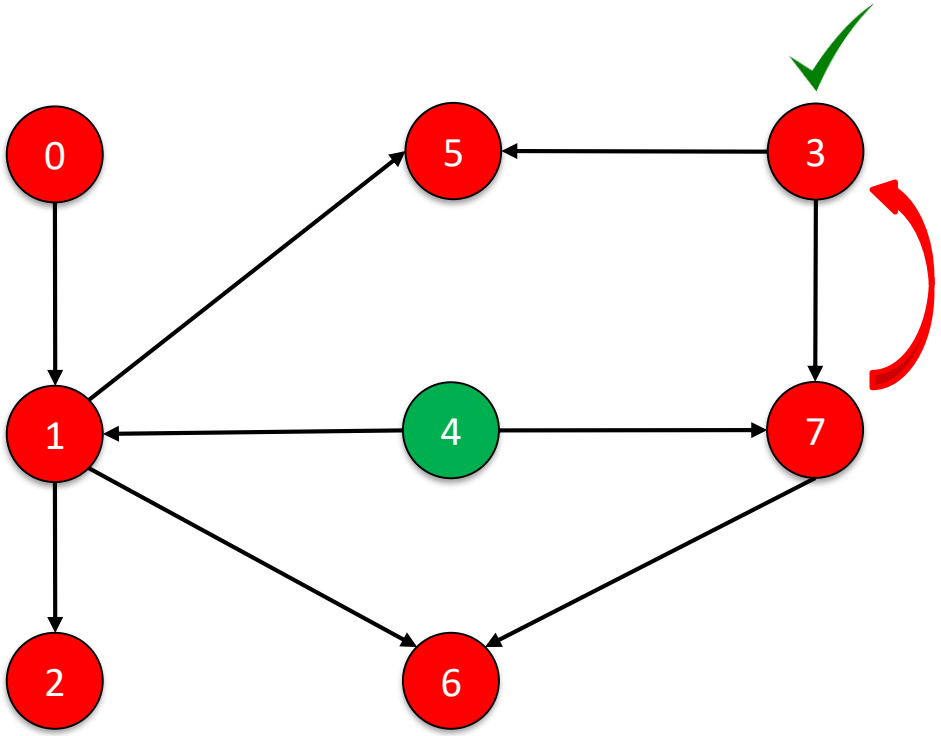
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	7	...	...

Bước 2: Lưu đỉnh 3 vào danh sách kết quả

Đỉnh 3 không còn đỉnh kề nào → lưu đỉnh 3 vào mảng kết quả.



visited

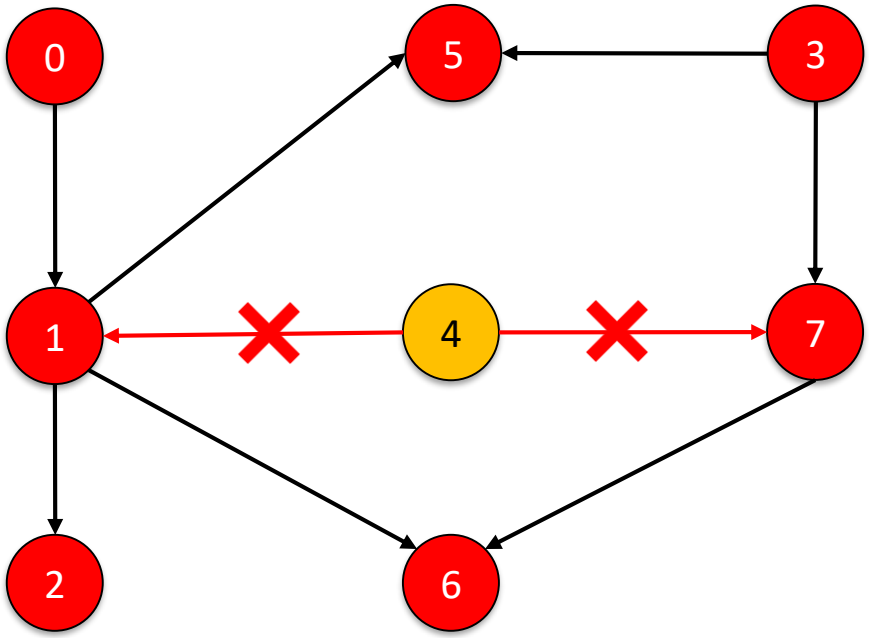
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	false	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	7	3	...

Bước 3: Chạy DFS cho đỉnh 4

Tìm đỉnh khác chưa viếng thăm. Gọi DFS đỉnh 4 → đi đến những đỉnh kề của đỉnh 4.



visited

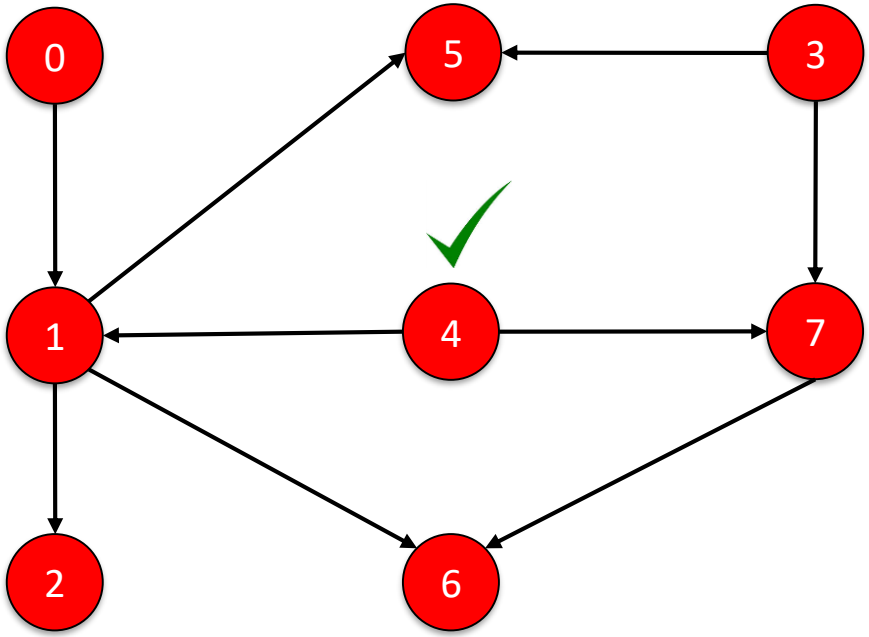
Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	true	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	7	3	...

Bước 3: Lưu đỉnh 4 vào danh sách kết quả

Đỉnh 4 không còn đỉnh kề nào → lưu đỉnh 4 vào mảng kết quả.



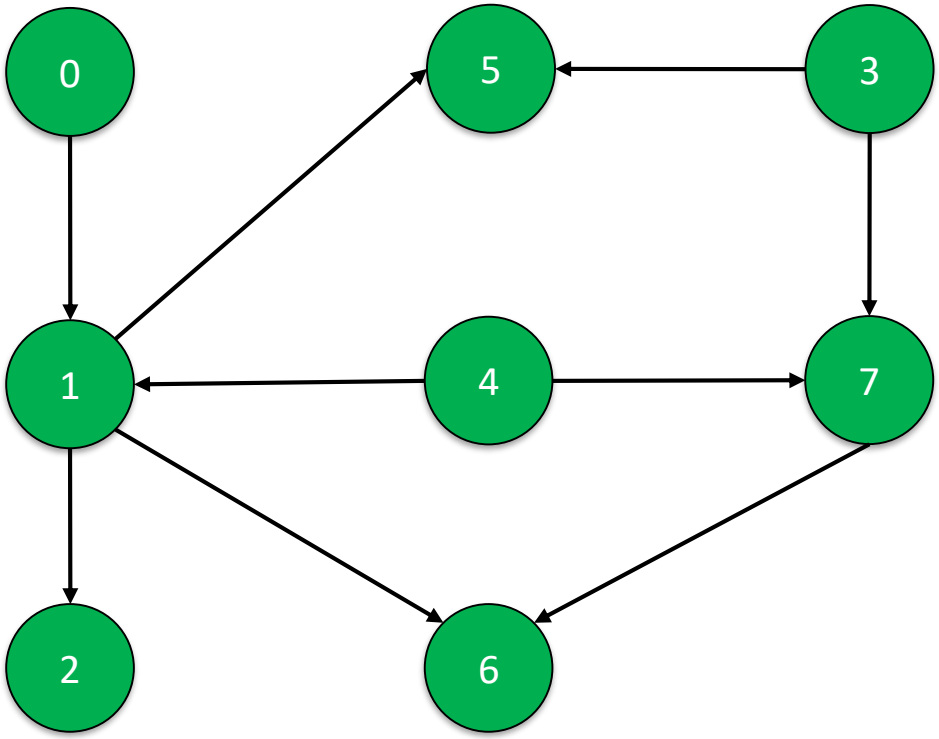
visited

Đỉnh	0	1	2	3	4	5	6	7
Trạng thái	true	true	true	true	true	true	true	true

result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	7	3	4

Kết quả bài toán



result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	2	5	6	1	0	7	3	4

Thứ tự Topo của đồ thị trên như sau: 4, 3, 7, 0, 1, 6, 5, 2.

Source Code Topological Sort DFS

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int V, E;
void dfs(int u, vector<vector<int>> &graph, vector<bool> &visited,
vector<int> &result) {
    visited[u] = true;
    for (int i = 0; i < graph[u].size(); i++) {
        int v = graph[u][i];
        if (!visited[v]) {
            dfs(v, graph, visited, result);
        }
    }
    result.push_back(u);
}
```

Source Code Topological Sort DFS

```
16. void topologicalSort(vector<vector<int>> &graph, vector<int> &result) {
17.     vector<bool> visited(V, false);
18.     for (int i = 0; i < V; i++) {
19.         if (!visited[i]) {
20.             dfs(i, graph, visited, result);
21.         }
22.     }
23.     reverse(result.begin(), result.end());
24. }
```

Source Code Topological Sort DFS

```cpp
int main() {
    vector<vector<int>> graph;
    vector<int> result;
    cin >> V >> E;
    graph.assign(V, vector<int>());
    for (int u, v, i = 0; i < E; i++) {
        cin >> u >> v;
        graph[u].push_back(v);
    }
    cout << "Topological Sort of graph:" << endl;
    topologicalSort(graph, result);
    for (int i = 0; i < V; i++) {
        cout << result[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Source Code Topological Sort DFS

```python
def dfs(u, graph, visited, result):
    visited[u] = True
    for v in graph[u]:
        if not visited[v]:
            dfs(v, graph, visited, result)
    result.append(u)

def topologicalSort(graph, result):
    visited = [False] * V
    for i in range(V):
        if not visited[i]:
            dfs(i, graph, visited, result)
    result.reverse()
```

Source Code Topological Sort DFS

```
13. if __name__ == "__main__":
14.     V, E = map(int, input().split())
15.     graph = [[] for i in range(V)]
16.     result = []
17.     for i in range(E):
18.         u, v = map(int, input().split())
19.         graph[u].append(v)
20.     topologicalSort(graph, result)
21.     print('Topological Sort of graph:')
22.     for i in range(V):
23.         print(result[i], end=' ')
```

Source Code Topological Sort DFS


```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;


public class Main {
    private static void dfs(int u, ArrayList<ArrayList<Integer>> graph, boolean[] visited, ArrayList<Integer> result) {
        visited[u] = true;
        for (int i = 0; i < graph.get(u).size(); i++) {
            int v = graph.get(u).get(i);
            if (!visited[v]) {
                dfs(v, graph, visited, result);
            }
        }
        result.add(u);
    }
```

Source Code Topological Sort DFS

```
15.     private static void topologicalSort(ArrayList<ArrayList<Integer>>
    graph, ArrayList<Integer> result) {
16.         int V = graph.size();
17.         boolean[] visited = new boolean[V];
18.         for (int i = 0; i < V; i++) {
19.             if (!visited[i]) {
20.                 dfs(i, graph, visited, result);
21.             }
22.         }
23.         Collections.reverse(result);
24.     }
```

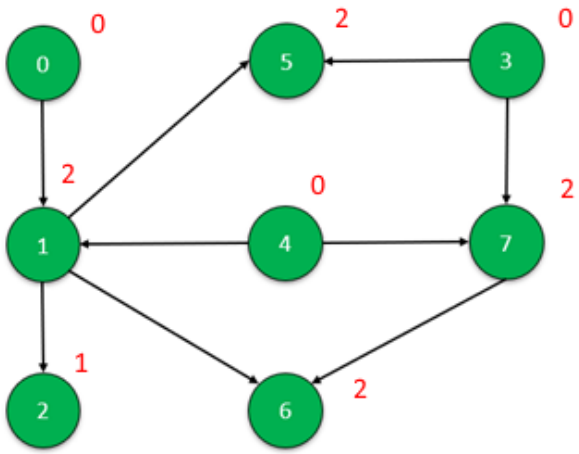
Source Code Topological Sort DFS

```
25.     public static void main(String[] args) {
26.         Scanner sc = new Scanner(System.in);
27.         int V = sc.nextInt();
28.         int E = sc.nextInt();
29.         ArrayList<ArrayList<Integer>> graph = new ArrayList<>();
30.         for (int i = 0; i < V; i++) {
31.             graph.add(new ArrayList<>());
32.         }
33.         for (int i = 0; i < E; i++) {
34.             int u = sc.nextInt();
35.             int v = sc.nextInt();
36.             graph.get(u).add(v);
37.         }
38.         ArrayList<Integer> result = new ArrayList<>();
39.         topologicalSort(graph, result);
40.         System.out.println("Topological Sort of graph:");
41.         for (int x: result) {
42.             System.out.printf("%d ", x);
43.         }
44.     }
45. }
```

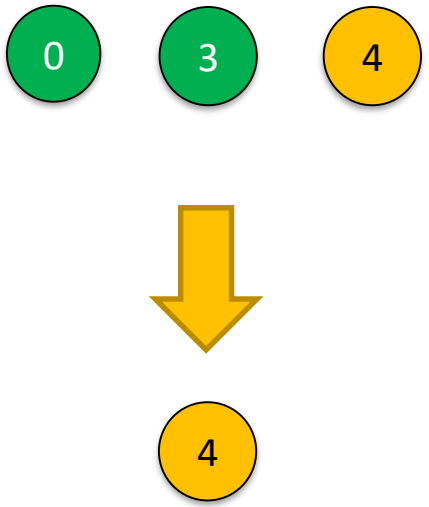
THUẬT TOÁN KAHN

Ý tưởng thuật toán Kahn

Tính bậc vào của từng đỉnh. Chọn những đỉnh có bậc vào là 0 lưu vào danh sách chờ xem xét.



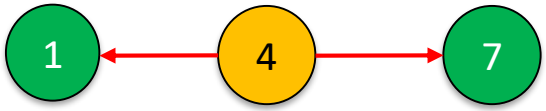
Chọn 1 đỉnh trong danh sách chờ ra xét.



Lưu đỉnh tìm được vào mảng kết quả.

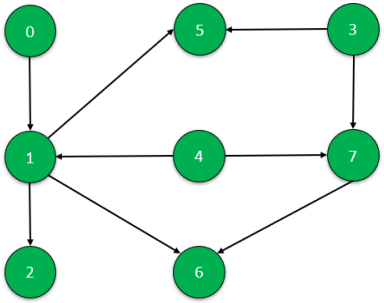
Vị trí	0	1	2	3
Đỉnh	4	...	...	...

Giảm bậc vào của các đỉnh kề với đỉnh đang xét. Thêm các đỉnh có bậc vào bằng 0 vào danh sách chờ.



**** Lưu ý: Trong đồ thị DAG có ít nhất 1 đỉnh có bậc vào bằng 0*

Bước 0: Chuẩn bị dữ liệu (1)



Chuyển danh sách cạnh kề vào CTDL **graph**.

Đỉnh	0	1	2	3	4	5	6	7
Đỉnh kề	1	2, 5, 6	…	5, 7	1, 7	…	…	6

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	**0**	**2**	**1**	**0**	**0**	**2**	**2**	**2**

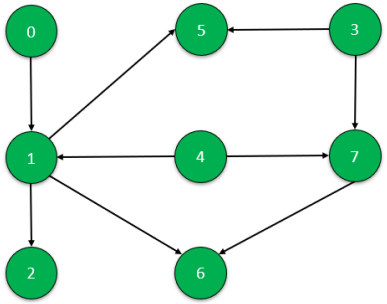
Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1	2	3	4	5	6	7
Đỉnh	…	…	…	…	…	…	…	…

Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	…	…	…	…	…	…	…	…

Bước 0: Chuẩn bị dữ liệu (2)



Chuyển danh sách cạnh kề vào CTDL **graph**.

Đỉnh	0	1	2	3	4	5	6	7
Đỉnh kề	1	2, 5, 6	...	5, 7	1, 7	...	...	6

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	0	2	1	0	0	2	2	2

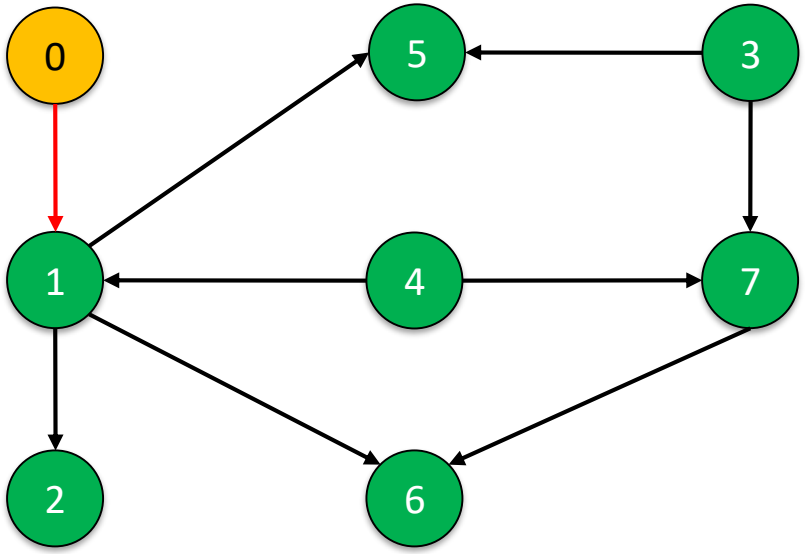
Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1	2
Đỉnh	**0**	**3**	**4**

Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	...	...	...	...	...	...	...	...

Bước 1: Chạy thuật toán lần 1 – xét đỉnh 0



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1	2
Đỉnh	**0**	3	4

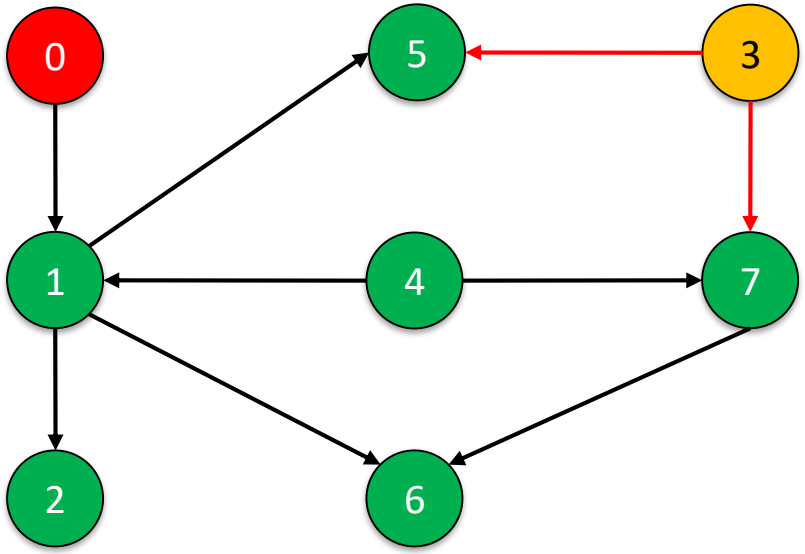
Mảng lưu kết quả **result**.

Vị trí	**0**	1	2	3	4	5	6	7
Đỉnh	**0**	...	...	...	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	**1**	2	3	4	5	6	7
Bậc vào	0	**2 → 1**	1	0	0	2	2	2

Bước 2: Chạy thuật toán lần 2 – xét đỉnh 3



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	1	2
Đỉnh	3	4

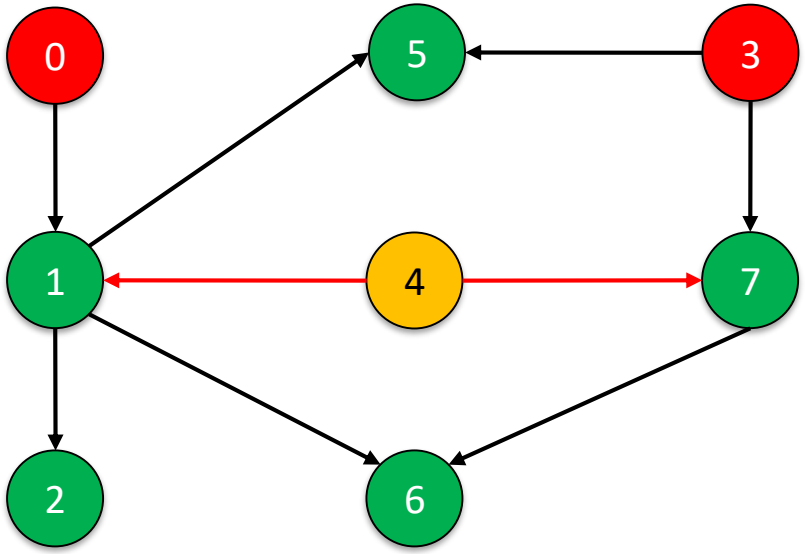
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	...	...	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	0	1	1	0	0	2 → 1	2	2 → 1

Bước 3: Chạy thuật toán lần 3 – xét đỉnh 4



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	2
Đỉnh	**4**

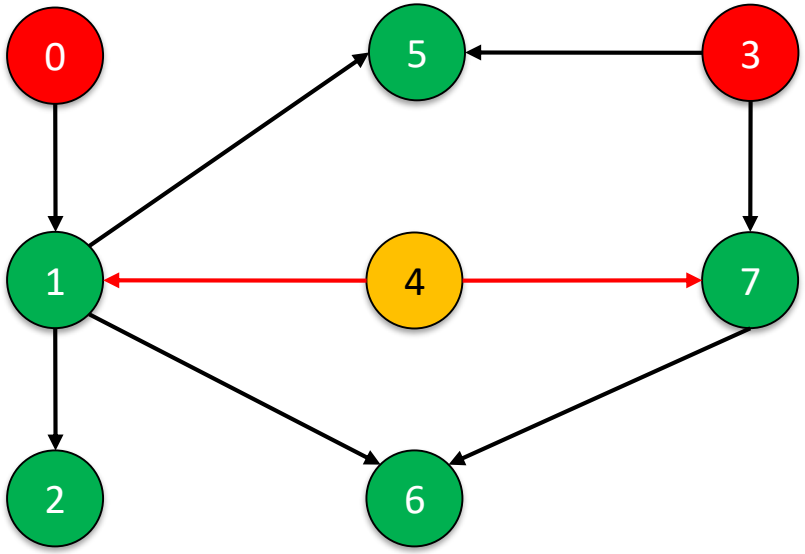
Mảng lưu kết quả **result**.

Vị trí	0	1	**2**	3	4	5	6	7
Đỉnh	0	3	**4**	...	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	**1**	2	3	4	5	6	**7**
Bậc vào	0	**1 → 0**	1	0	0	1	2	**1 → 0**

Bước 3: Chạy thuật toán lần 3 – xét đỉnh 4



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	**1**
Đỉnh	**1**	**7**

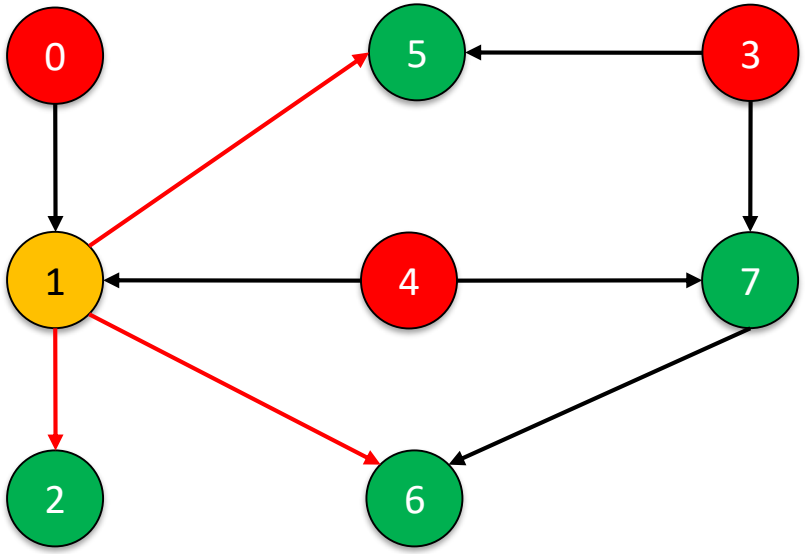
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	...	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	**1**	2	3	4	5	6	**7**
Bậc vào	0	**0**	1	0	0	1	2	**0**

Bước 4: Chạy thuật toán lần 4 – xét đỉnh 1



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1
Đỉnh	**1**	7

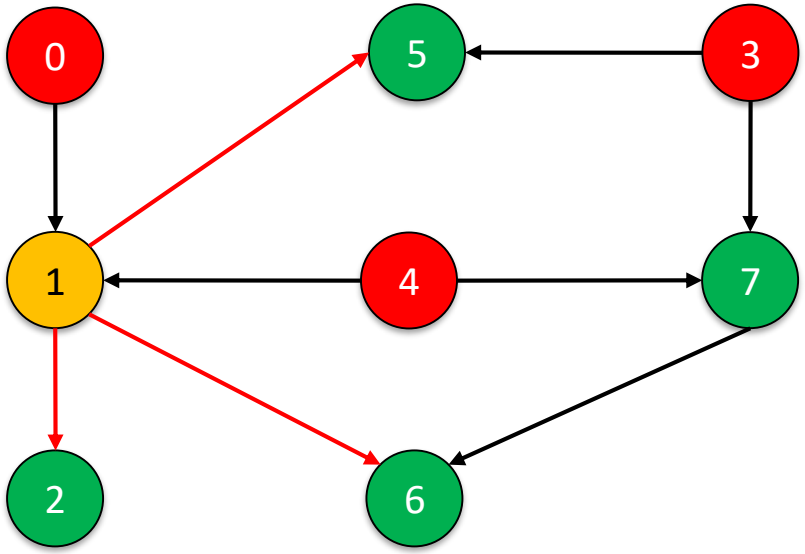
Mảng lưu kết quả **result**.

Vị trí	0	1	2	**3**	4	5	6	7
Đỉnh	0	3	4	**1**	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	**2**	3	4	**5**	**6**	7
Bậc vào	0	0	**1 → 0**	0	0	**1 → 0**	**2 → 1**	0

Bước 4: Chạy thuật toán lần 4 – xét đỉnh 1



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1	2
Đỉnh	7	2	5

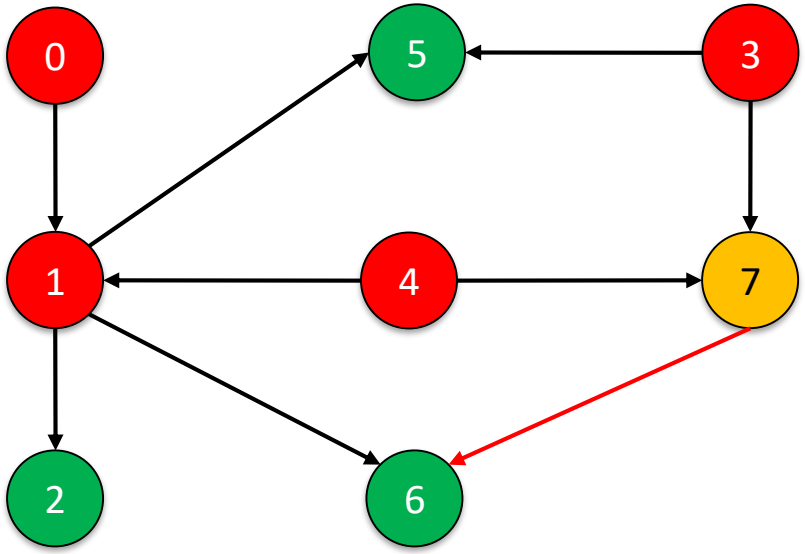
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	1	...	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	**2**	3	4	**5**	6	7
Bậc vào	0	0	**0**	0	0	**0**	1	0

Bước 5: Chạy thuật toán lần 5 – xét đỉnh 7



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	0	1	2
Đỉnh	**7**	2	5

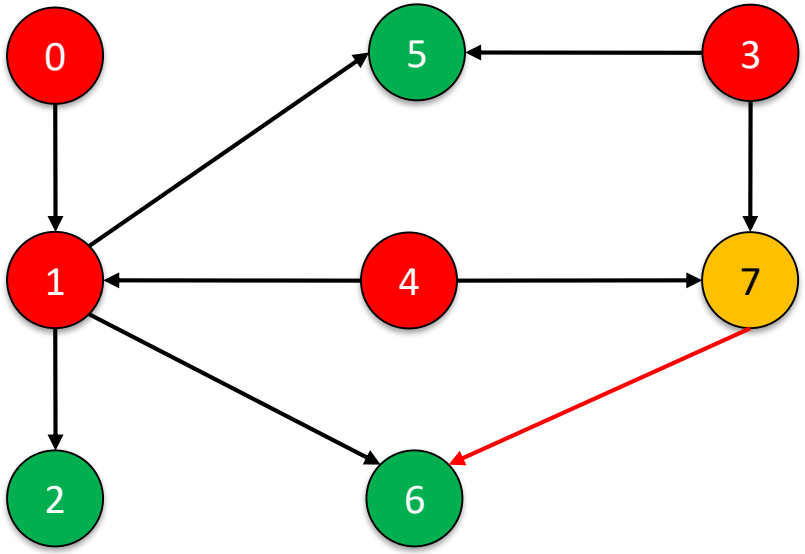
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	**4**	5	6	7
Đỉnh	0	3	4	1	**7**	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	**6**	7
Bậc vào	0	0	0	0	0	0	**1 → 0**	0

Bước 5: Chạy thuật toán lần 5 – xét đỉnh 7



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	1	2	3
Đỉnh	2	5	6

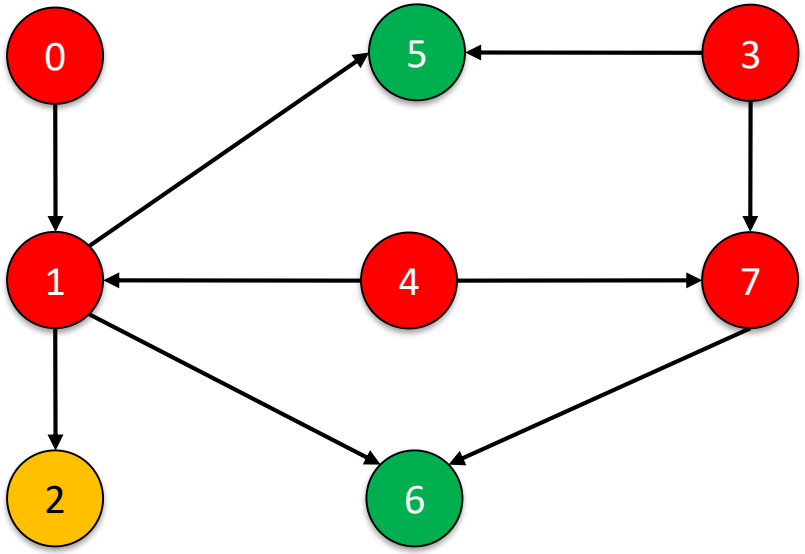
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	1	7	...	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	**6**	7
Bậc vào	0	0	0	0	0	0	**0**	0

Bước 6: Chạy thuật toán lần 6 – xét đỉnh 2



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	1	2	3
Đỉnh	2	5	6

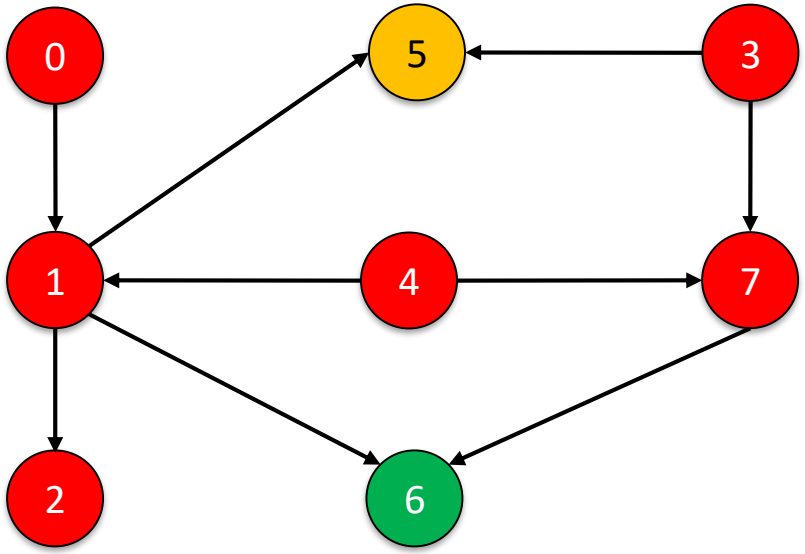
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	1	7	2	...	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	0	0	0	0	0	0	0	0

Bước 7: Chạy thuật toán lần 7 – xét đỉnh 5



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	2	3
Đỉnh	**5**	6

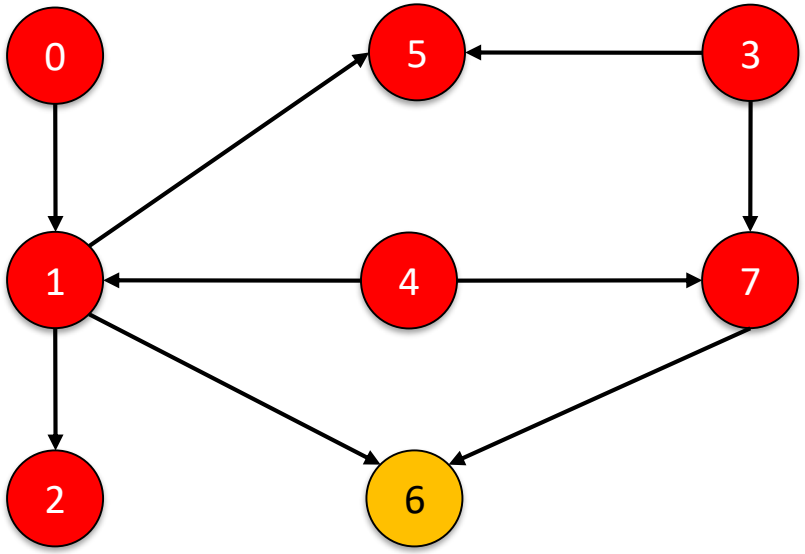
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	**6**	7
Đỉnh	0	3	4	1	7	2	**5**	...

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	0	0	0	0	0	0	0	0

Bước 8: Chạy thuật toán lần 8 – xét đỉnh 6



Hàng đợi chứa đỉnh có bậc vào bằng 0 **zero_indegree.**

	3
Đỉnh	6

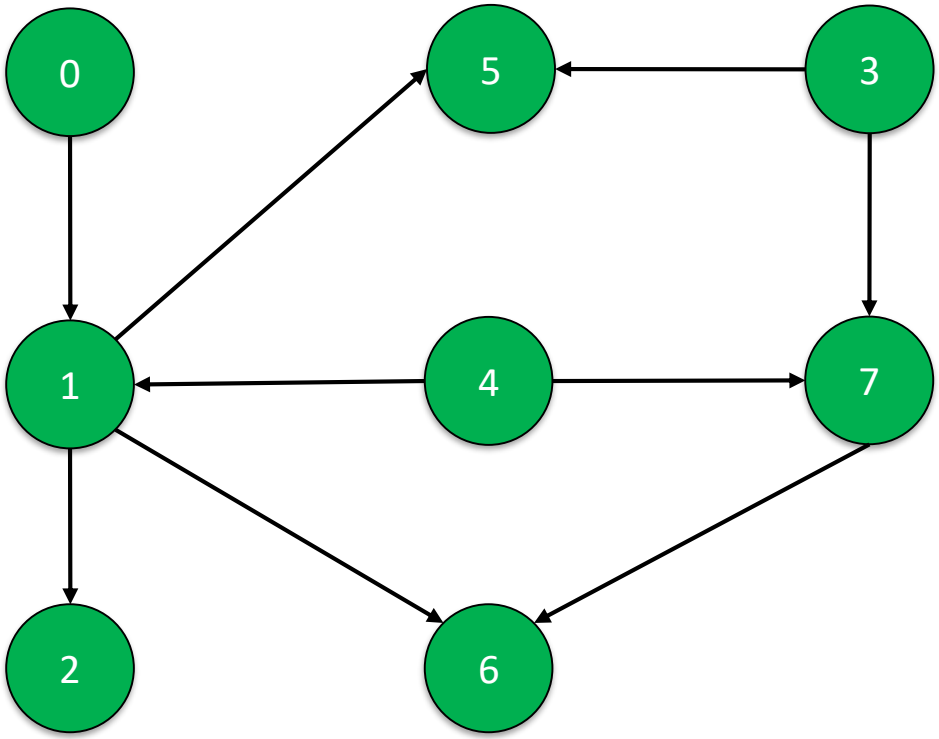
Mảng lưu kết quả **result**.

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	1	7	2	5	6

Mảng bậc vào của đỉnh **indegree.**

Đỉnh	0	1	2	3	4	5	6	7
Bậc vào	0	0	0	0	0	0	0	0

Kết quả bài toán



result

Vị trí	0	1	2	3	4	5	6	7
Đỉnh	0	3	4	1	7	2	5	6

Thứ tự Topo của đồ thị trên như sau: 0, 3, 4, 1, 7, 2, 5, 6.

Source Code Kahn's Algorithm

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int V, E;
bool topologicalSort(vector<vector<int>> &graph, vector<int> &result){
    vector<int> indegree(V, 0);
    queue<int> zero_indegree;
    for (int u = 0; u < V; u++) {
        vector<int>::iterator it;
        for (it = graph[u].begin(); it != graph[u].end(); it++)
            indegree[*it]++;
    }
    for (int i = 0; i < V; i++) {
        if (indegree[i] == 0) {
            zero_indegree.push(i);
        }
    }
```

Source Code Kahn's Algorithm

```
19.       while (!zero_indegree.empty()) {
20.           int u = zero_indegree.front();
21.           zero_indegree.pop();
22.           result.push_back(u);
23.           vector<int>::iterator it;
24.           for (it = graph[u].begin(); it != graph[u].end(); it++) {
25.               indegree[*it]--;
26.               if (indegree[*it] == 0) {
27.                   zero_indegree.push(*it);
28.               }
29.           }
30.       }
31.       for (int i = 0; i < V; i++) {
32.           if (indegree[i] != 0)
33.               return false;
34.       }
35.       return true;
36.   }
```

Source Code Kahn's Algorithm

```cpp
int main() {
    vector<vector<int>> graph;
    vector<int> result;
    cin >> V >> E;
    graph.assign(V, vector<int>());
    for (int u, v, i = 0; i < E; i++) {
        cin >> u >> v;
        graph[u].push_back(v);
    }
    cout << "Topological Sort of graph:" << endl;
    if (topologicalSort(graph, result) == true) {
        for (int i = 0; i < result.size(); i++)
            cout << result[i] << " ";
    }
    else
        cout << "No result";
    return 0;
}
```

Source Code Kahn's Algorithm

```
1.  import queue


2.  def topologicalSort(graph, result):
3.      indegree = [0] * V
4.      zero_indegree = queue.Queue()
5.      for u in range(V):
6.          for v in graph[u]:
7.              indegree[v] += 1
8.      for i in range(V):
9.          if indegree[i] == 0:
10.             zero_indegree.put(i)
```

Source Code Kahn's Algorithm

```
11.     while not zero_indegree.empty():
12.         u = zero_indegree.get()
13.         result.append(u)
14.         for v in graph[u]:
15.             indegree[v] -= 1
16.             if indegree[v] == 0:
17.                 zero_indegree.put(v)
18.     for i in range(V):
19.         if indegree[i] != 0:
20.             return False
21.     return True
```

Source Code Kahn's Algorithm

```
23. if __name__ == "__main__":
24.     V, E = map(int, input().split())
25.     graph = [[] for i in range(V)]
26.     result = []
27.     for i in range(E):
28.         u, v = map(int, input().split())
29.         graph[u].append(v)
30.     if (topologicalSort(graph, result)):
31.         print('Topological Sort of graph:')
32.         for i in range(V):
33.             print(result[i], end = ' ')
34.     else:
35.         print("No result")
```

Source Code Kahn's Algorithm

```
1.  import java.util.*;
2.  public class Main {
3.      private static boolean topologicalSort(ArrayList<ArrayList<Integer>>
    graph, ArrayList<Integer> result) {
4.          int V = graph.size();
5.          int[] indegree = new int[V];
6.          Queue<Integer> zeroIndegree = new LinkedList<>();
7.          for (int u = 0; u < V; u++) {
8.              for (int v : graph.get(u)) {
9.                  indegree[v]++;
10.             }
11.         }
12.         for (int i = 0; i < V; i++) {
13.             if (indegree[i] == 0) {
14.                 zeroIndegree.add(i);
15.             }
16.         }
```

Source Code Kahn's Algorithm

```
17.         while (! zeroIndegree.isEmpty()) {
18.             int u = zeroIndegree.poll();
19.             result.add(u);
20.             for (int v : graph.get(u)) {
21.                 indegree[v]--;
22.                 if (indegree[v] == 0) {
23.                     zeroIndegree.add(v);
24.                 }
25.             }
26.         }
27.         for (int i = 0; i < V; i++) {
28.             if (indegree[i] != 0) {
29.                 return false;
30.             }
31.         }
32.         return true;
33.     }
```

Source Code Kahn's Algorithm

```
34.     public static void main(String[] args) {
35.         Scanner sc = new Scanner(System.in);
36.         int V = sc.nextInt();
37.         int E = sc.nextInt();
38.         ArrayList<ArrayList<Integer>> graph = new ArrayList<>();
39.         for (int i = 0; i < V; i++) {
40.             graph.add(new ArrayList<>());
41.         }
42.         for (int i = 0; i < E; i++) {
43.             int u = sc.nextInt();
44.             int v = sc.nextInt();
45.             graph.get(u).add(v);
46.         }
```

Source Code Kahn's Algorithm

```
47.             ArrayList<Integer> result = new ArrayList<>();
48.             if (topologicalSort(graph, result)) {
49.                 System.out.println("Topological Sort of graph:");
50.                 for (int x: result) {
51.                     System.out.printf("%d ", x);
52.                 }
53.             }
54.             else {
55.                 System.out.println("No result");
56.             }
57.         }
58. }
```

Hỏi đáp